PCクラスタワークショップ
HPC-OSS部会セッション

AMD ROCmソフトウェアについて

日本AMD株式会社
大原久樹 (Hisaki.Ohara@amd.com)

AMD
together we advance_

[Public]

AMD ROCm

Optimized AI software stack

AI Models and **Algorithms**

PyTorch + Other Frameworks

AI Ecosystem optimized for AMD

Libraries

Compilers and **Tools**

Runtime

AMD ROCm | A proven software stack

AMD Instinct™ GPU

AMD INSTINCT | Leadership performance

- Commitment to **Open-Source**
- **No Code Change** Execution
- Optimized for **Generative AI**

Use of third party marks/logos/products is for informational purposes only and no endorsement of or by AMD is intended or implied GD-83

AMD
together we advance_

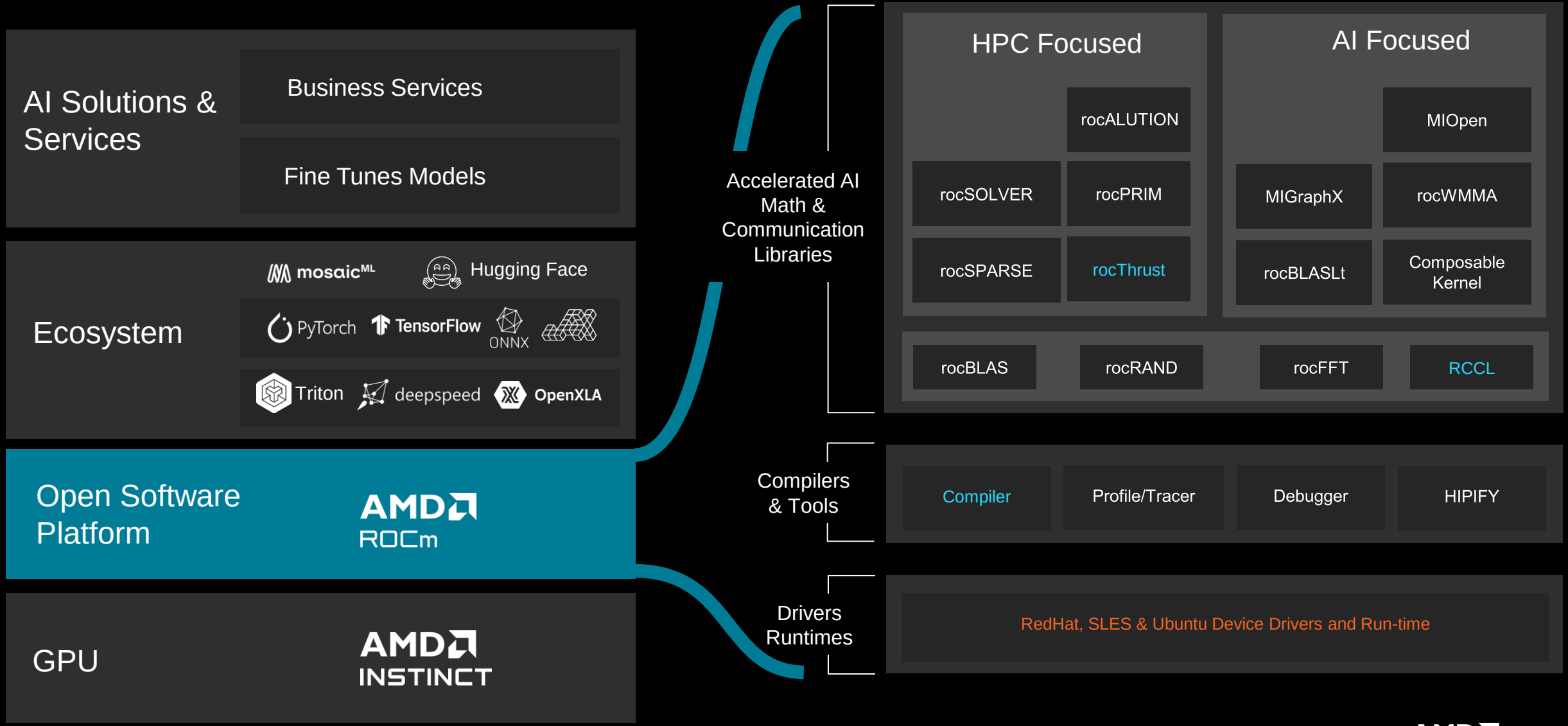
[Public]

MIT/BSD License

Apache License

GPL License

AMD ROCm™ Software Stack



Use of third party marks/logos/products is for informational purposes only and no endorsement of or by AMD is intended or implied GD-83

AMD
together we advance_

[Public]

What is Heterogeneous-Compute Interface (HIP)?

- C++ runtime API and kernel language with CUDA-like APIs
- HIP allows developers to create portable applications for AMD and NVIDIA GPUs
- *cudaFunction* style functions expressed as *hipFunction*

CUDA	HIP
cudaMalloc (&d_x, N*sizeof(float));	hipMalloc (&d_x, N*sizeof(float));
cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);	hipMemcpy(d_x, x, N*sizeof(float), hipMemcpyHostToDevice);
cudaDeviceSynchronize();	hipDeviceSynchronize();

AMD
together we advance_

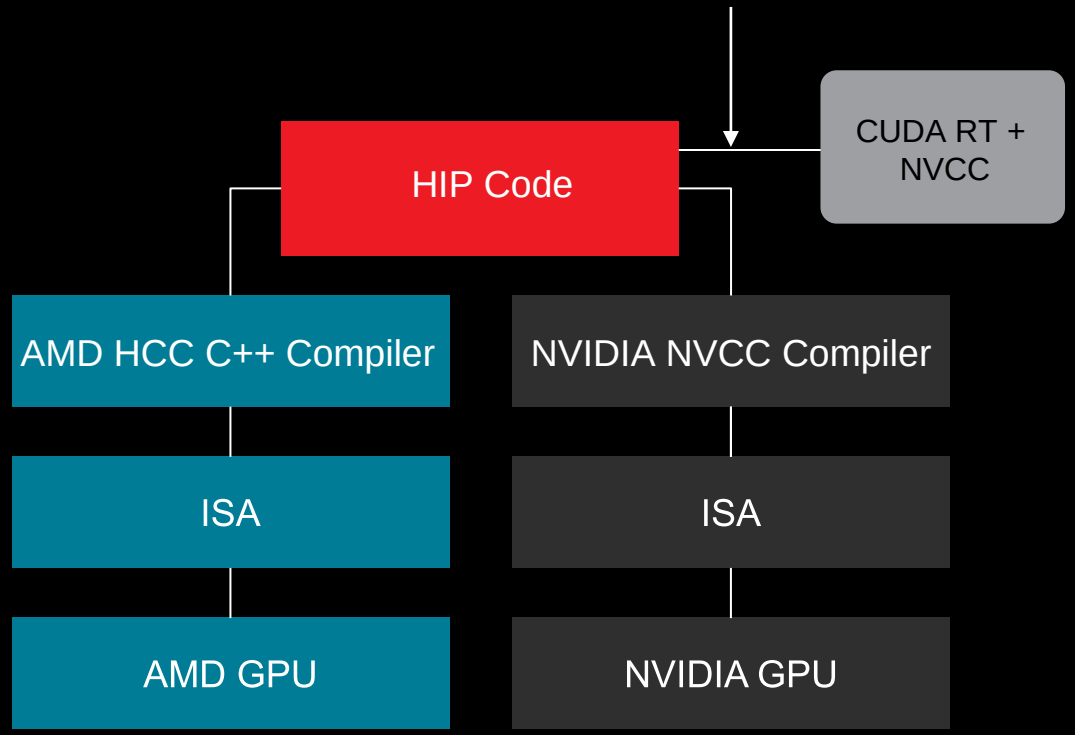
[Public]

HIP – Seamless Porting from CUDA to ROCm™ SW

Code Conversion Workflow Diagram

Simple CUDA to HIP Conversion Tool

- Hipify-perl: simple and fast
- Hipify-clang: high-quality translation



Kernels Comparison

CUDA Vector Add

```
__global__ void add(int n,
                    double *x,
                    double *y){
  int index = blockIdx.x * blockDim.x
              + threadIdx.x;
  int stride = blockDim.x * gridDim.x;

  for (int i = index; i < n; i +=
stride){
    y[i] = x[i] + y[i];
  }
}
```

HIP Vector Add

```
__global__ void add(int n,
                    double *x,
                    double *y){
  int index = blockIdx.x * blockDim.x
              + threadIdx.x;
  int stride = blockDim.x * gridDim.x;

  for (int i = index; i < n; i += stride){
    y[i] = x[i] + y[i];
  }
}
```

Kernels are syntactically the same

AMD
together we advance_

[Public]

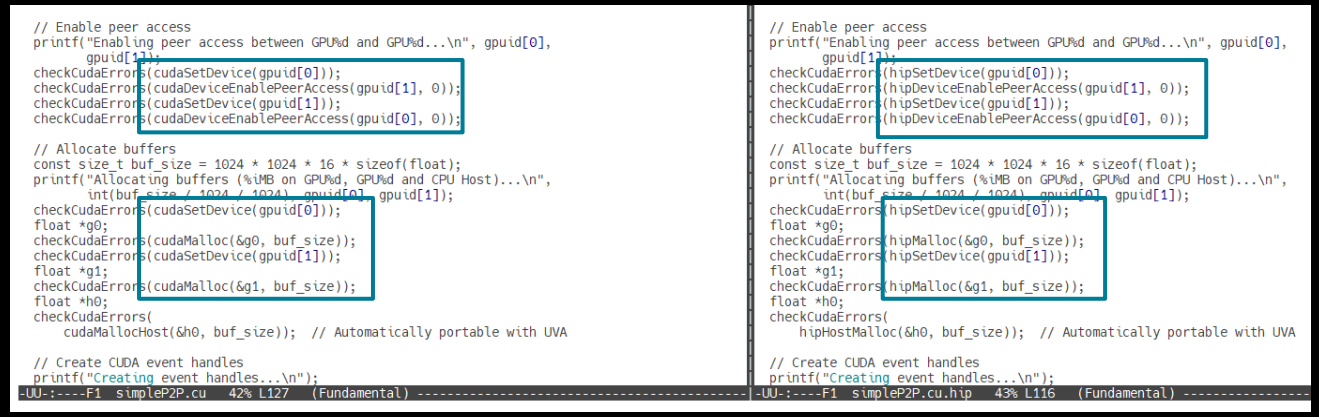
HIPIFY - Automatic CUDA to HIP

A set of tools that translate CUDA code into HIP C++

- **hipify-perl**
  - Simple string replacement technique
  - Recommended for quick scans of projects
- **hipify-clang**
  - Requires clang compiler to parse source files
  - High quality translation

Limitations

- Inline NVIDIA PTX assembly
- CUDA intrinsic functions
- Hard-coded dependencies on warp size, shared memory size
- Code geared toward the limited size of register file on NVIDIA hardware
- Unsupported libraries, e.g., CUTLASS to CK



```
// Enable peer access
printf("Enabling peer access between GPU%d and GPU%d...\n", gpuid[0],
       gpuid[1]);
checkCudaErrors(cudaSetDevice(gpuid[0]));
checkCudaErrors(cudaDeviceEnablePeerAccess(gpuid[1], 0));
checkCudaErrors(cudaSetDevice(gpuid[1]));
checkCudaErrors(cudaDeviceEnablePeerAccess(gpuid[0], 0));

// Allocate buffers
const size_t buf_size = 1024 * 1024 * 16 * sizeof(float);
printf("Allocating buffers (%iMB on GPU%d, GPU%d and CPU Host)...\n",
       int(buf_size / 1024 / 1024), gpuid[0], gpuid[1]);
checkCudaErrors(cudaSetDevice(gpuid[0]));
float *g0;
checkCudaErrors(cudaMalloc(&g0, buf_size));
checkCudaErrors(cudaSetDevice(gpuid[1]));
float *g1;
checkCudaErrors(cudaMalloc(&g1, buf_size));
float *h0;
checkCudaErrors(
    cudaMallocHost(&h0, buf_size));  // Automatically portable with UVA

// Create CUDA event handles
printf("Creating event handles...\n");
-UU-:----F1  simpleP2P.cu    42% L127   (Fundamental) ----------
```

```
// Enable peer access
printf("Enabling peer access between GPU%d and GPU%d...\n", gpuid[0],
       gpuid[1]);
checkCudaErrors(hipSetDevice(gpuid[0]));
checkCudaErrors(hipDeviceEnablePeerAccess(gpuid[1], 0));
checkCudaErrors(hipSetDevice(gpuid[1]));
checkCudaErrors(hipDeviceEnablePeerAccess(gpuid[0], 0));

// Allocate buffers
const size_t buf_size = 1024 * 1024 * 16 * sizeof(float);
printf("Allocating buffers (%iMB on GPU%d, GPU%d and CPU Host)...\n",
       int(buf_size / 1024 / 1024), gpuid[0], gpuid[1]);
checkCudaErrors(hipSetDevice(gpuid[0]));
float *g0;
checkCudaErrors(hipMalloc(&g0, buf_size));
checkCudaErrors(hipSetDevice(gpuid[1]));
float *g1;
checkCudaErrors(hipMalloc(&g1, buf_size));
float *h0;
checkCudaErrors(
    hipHostMalloc(&h0, buf_size));  // Automatically portable with UVA

// Create CUDA event handles
printf("Creating event handles...\n");
-UU-:----F1  simpleP2P.cu.hip   43% L116   (Fundamental) ----------
```


AMD
together we advance_

[Public]

Example: Qulacs (quantum simulator) Porting to HIP

qulacs / qulacs

Type / to search

Code | Issues 20 | Pull requests | Discussions | Actions | Projects | Wiki | Security | Insights

Adding ROCm support for AMD GPUs #638

Merged KowerKoint merged 33 commits into `qulacs:main` from `hisohara:rocm` last month

Conversation 26 | Commits 33 | Checks 10 | Files changed 15

hisohara commented on Apr 26 — Contributor

This PR adds the support of ROCm with HIP for AMD GPUs. CUDA codes are converted to HIP API. Per the past discussion with maintainer, `#ifdef __HIP_PLATFORM_AMD__` is used for HIP API. This HIP region is called only when HIP compiler is used. No change on CUDA codes.

To compile for ROCm, `USE_HIP=Yes` needs to be added on `USE_GPU=Yes` as follows:
`$ USE_GPU=Yes USE_HIP=Yes pip install .`

Tested hardware environment:
MI250, MI300A

Tested software environment:
ROCm 6.0.2 and ROCm 6.1

References

- AMD ROCm documentation
- MI250 hardware information
- MI300A hardware information

1 | 1

129 src/gpusim/update_ops_single.cu

```
@@ -1,8 +1,6 @@
- #include <cuComplex.h>
+ #include "gpu_wrapping.h"
- #include "cuda_runtime.h"
- #include "device_launch_parameters.h"
- // #include "util.h"
+ //#include "util.h"
  #include <assert.h>
  #include "update_ops_cuda.h"
@@ -79,10 +77,10 @@ __device__ void single_qubit_dense_matrix_gate_device(
          basis1 = basis0 ^ (1ULL << target_qubit_index);
          tmp = state_gpu[basis0];
-         state_gpu[basis0] = cuCadd(cuCmul(matrix_const_gpu[0], tmp),
-             cuCmul(matrix_const_gpu[1], state_gpu[basis1]));
-         state_gpu[basis1] = cuCadd(cuCmul(matrix_const_gpu[2], tmp),
-             cuCmul(matrix_const_gpu[3], state_gpu[basis1]));
+         state_gpu[basis0] = gpuCadd(gpuCmul(matrix_const_gpu[0], tmp),
+             gpuCmul(matrix_const_gpu[1], state_gpu[basis1]));
+         state_gpu[basis1] = gpuCadd(gpuCmul(matrix_const_gpu[2], tmp),
+             gpuCmul(matrix_const_gpu[3], state_gpu[basis1]));
```

https://github.com/qulacs/qulacs/pull/638

AMD
together we advance_

[Public]

Compilers – two major strands

Primary Support

- LLVM™ based
  - ROCmCC provides support for both HIP and OpenMP®
    - hipcc -- /opt/rocm/bin
    - amdclang -- /opt/rocm/llvm/bin
  - AOMP: AMD OpenMP® research compiler for prototyping new features for ROCmCC

Functionality Only

- GCC based -- GNU Compilers
  - Provide offloading support to AMD GPUs (OpenMP®, OpenACC)
    - GCC 11 added offloading for the AMD MI100 GPUs
    - GCC 13 adds support for the AMD MI200 GPU series.
    - OG -- The devel/omp/gcc-13 (OG13) is the development
  - https://gcc.gnu.org/wiki/Offloading

AMD
together we advance_

[Public]

Contribution to LLVM

```
hisohara:~/Projects/ROCm/llvm-project$ git log --since="2023-0626" --pretty=format:'%ae’|
awk -F '@' '{print $2}'|sort|uniq -c|sort -nr|head -15
   6261 gmail.com
   4219 google.com
   2958 users.noreply.github.com
   2415 amd.com
   1939 arm.com
   1897 redhat.com
   1181 apple.com
   1146 sifive.com
    921 redking.me.uk
    894 outlook.com
    863 maskray.me
    744 intel.com
    672 nvidia.com
    517 fhahn.com
    464 igalia.com
```


AMD
together we advance_

[Public]

Pragma-based Languages

- OpenMP® – primary supported option for AMD GPUs
  - Implemented through LLVM™
- OpenACC – Supported through Cray, LLVM™ (CLACC) and GCC compilers
  - Also available are source-to-source translation tools from OpenACC to OpenMP (Intel® and CLACC)
  - Not as well supported as other options
- Standard Language support – mostly a future option
  - Long-time discussed “for_each”, “do_concurrent”, “for_all”,
  - Parallel C++ through parallel “execution policies”

AMD
together we advance_

[Public]

Higher Level Performance Portability Frameworks

- Kokkos – Sandia National Lab (SNL) C++ performance portable programming model
  - The Kokkos team has agressively developed support for AMD GPUs via a HIP backend
  - Kokkos handles many of the unique attributes of the AMD GPUs for you
  - Parts being integrated into the C++ standard

- RAJA – Lawrence Livermore National Lab (LLNL) C++ performance portability layer
  - Modular in structure with separation of compute and data management
  - Supports AMD GPUs
  - Key kernel patterns have been optimized by AMD

- Advantages of Performance Portability Frameworks
  - True single-source application code (at least if you restrict yourself to C++)
  - Many of these frameworks support both CPUs and GPUs

AMD
together we advance_

[Public]

Expanding Software Developer Ecosystem

Hugging Face

62,000+ CI/CD tests running nightly
Fully integrated optimum library

PyTorch

From ‘port-to’ to ‘develop-on’
with latest platforms

- Tensor Flow
- Dynamo Instructor
- JAX
- OpenAI Triton
- ONNX Runtime
- OpenXLA
- DeepSpeed
- MLIR | IREE

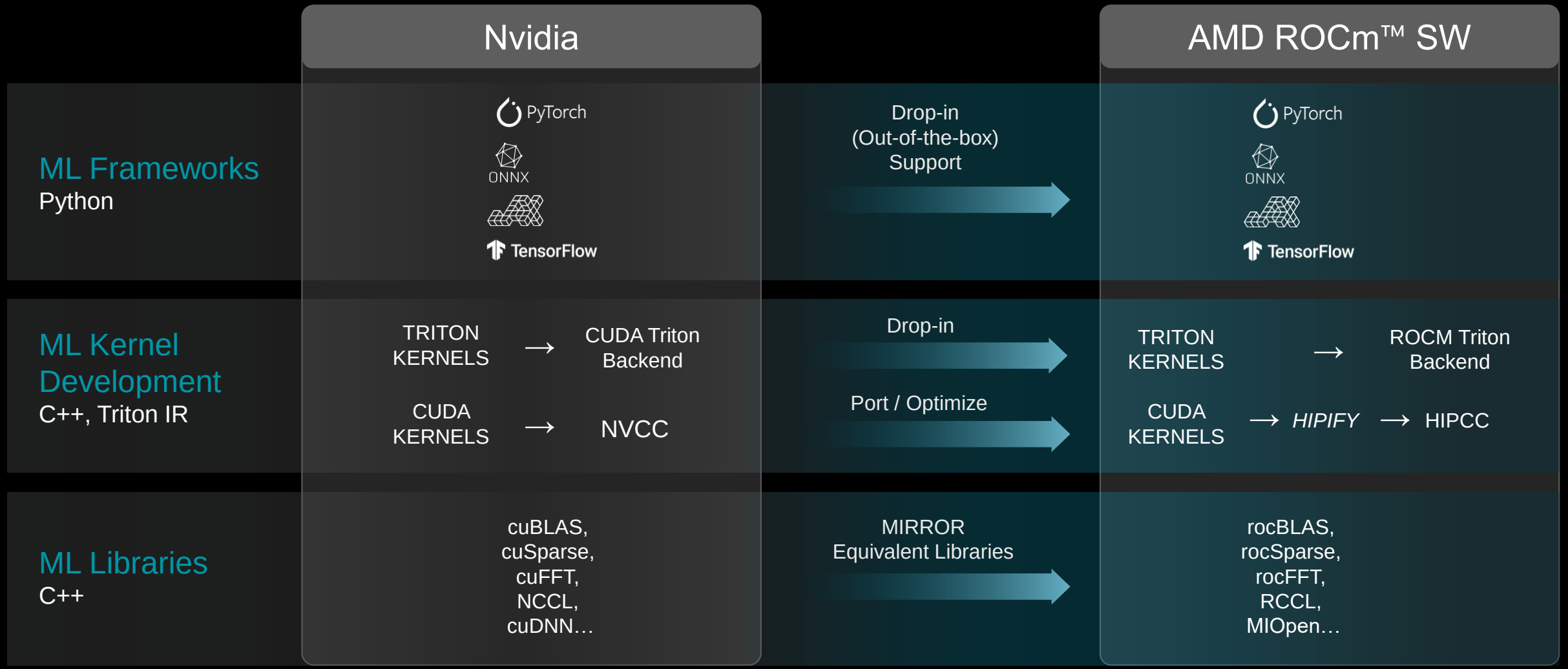
Increasing open-source
contributions and expanding footprint

Use of third party marks/logos/products is for informational purposes only and no endorsement of or by AMD is intended or implied GD-83

AMD
together we advance_

[Public]

Transitioning AI Workloads to AMD GPUs



Use of third party marks/logos/products is for informational purposes only and no endorsement of or by AMD is intended or implied GD-83

AMD
together we advance_

[Public]

AMD + 🤗: LLM Out-of-the-Box Acceleration with AMD GPU

1. Supports all transformers models and tasks
2. Zero code change even keep “cuda”
3. Latest LLM optimization features

```
import torch
# Support all transformers models and tasks on AMD
from transformers import AutoTokenizer, AutoModelForCausalLM

model_id = "TheBloke/Llama-2-7B-fp16"
tokenizer = AutoTokenizer.from_pretrained(model_id)
# Zero code change between Nvidia and AMD
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

with torch.device(device):
# Latest innovations and features are available (flash_attention_2)
    model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype=torch.float16, attn_implementation="flash_attention_2")
    token_ids = tokenizer([”…"], padding=False, return_tensors="pt").to(device)
    generated_ids = model.generate(**token_ids, max_new_tokens=50, early_stopping=True)
print(tokenizer.batch_decode(generated_ids))
```

1 2 3

https://huggingface.co/blog/huggingface-and-optimum-amd

AMD
together we advance_


AMD